15.–18. 09. 2008
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Nähkästchenplauderei

Erfahrungen mit JBoss Seam

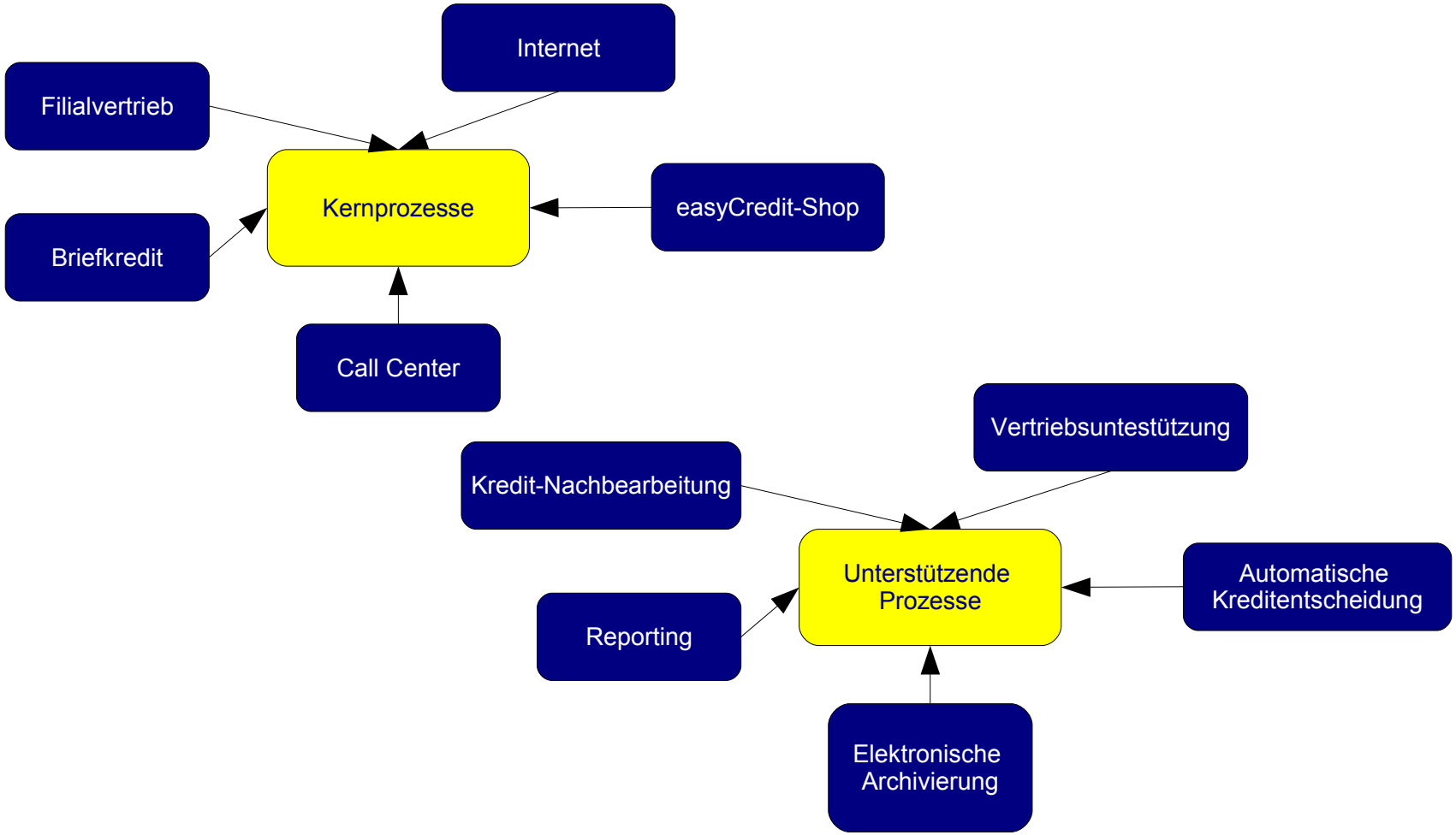
Werner Eberling / Francis Pouatcha

MATHEMA Software GmbH / adorsys Ltd. & Co. KG

Agenda

- Ausgangssituation
- Erste Seam-Erfahrungen
- Architektur
  - Frontend
  - Backend
- Entwicklungsumgebung
- Lessons learned

easyCredit: Das Geschäft



Ausgangsituation

- Erfolgreiche easyCredit-Anwendung auf Basis von J2EE (ANT, Struts, EJB2.1, OJB auf JBoss)
  - Entwicklungsstart Januar 2003, Produktionseinführung Dezember 2003.
  - Anwendung versorgt ca. 1000 Volks- und Raiffeisenbanken mit Verbraucherkredite.
- Ab 2005 neue Bausteine mit Spring/Hibernate

Ausgangssituation

- Januar 2007: Entwurf einer neuen mehrländerfähige Version der Anwendung.
- Neue fachliche Prozesse.
- Technologisch: Maven2, EJB3, JPA, JSF, SEAM, Embedded EJB3.

Das erste mal Seam

- Erste Seam-Erfahrungen im Rahmen eines Migrationsprojektes
  - Ablösung einer existierenden Internetanwendung
  - EJB 2.x Beans adaptiert durch „dünne“ EJB3 Schicht
  - Integration von EJB3 ins Frontend erstmals über Seam
- Produktiv seit Q1/2007

Die Versprechungen

- Einfachere Integration des Backends
  - Direkter Zugriff auf EJB3-Beans
  - Auflösung der klassischen Mehrschicht-Architektur
- Wegfall von Modell-Mappings
  - Direkter Zugriff auf JPA-Entitäten
- Höhere Usability durch zusätzliche Scopes
  - Spez: Conversation-Scope

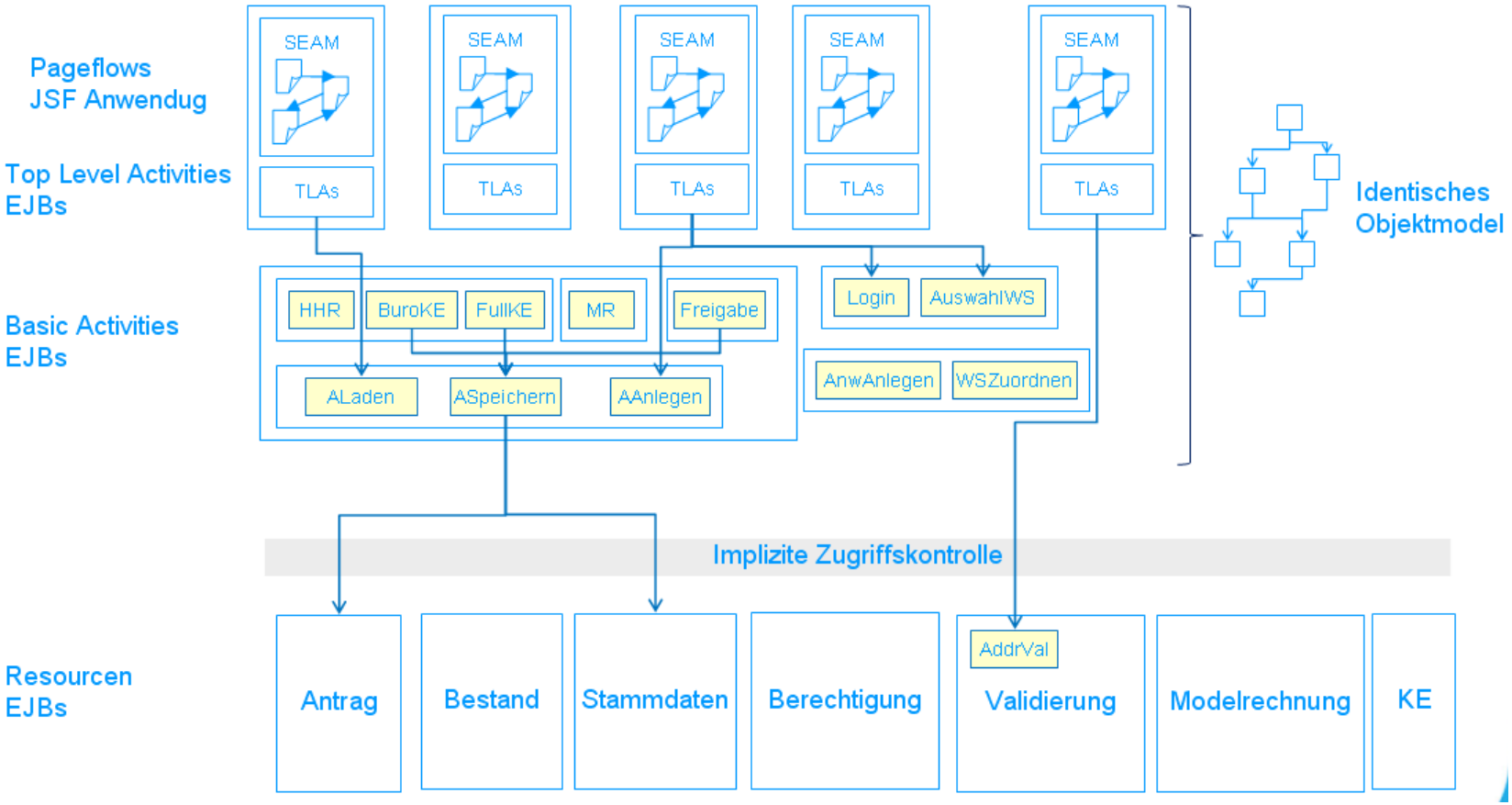
Die Versprechungen

- Definition von Pre-Actions
- Erweiterung des Dependency-Injection-Modells
  - In/Outjection (Bijection)

Nach dem ersten mal Seam

- Nach ersten Erfahrungen
  - Entscheidung für SEAM in der Breite
  - Durchgängiger Einsatz des neuen Ansatzes
  - Ausarbeitung einer tragfähigen Architektur

Architektur-Überblick



Top-Aktiv im Frontend

- Abbildung des JSF-Pageflows auf „Top-Level-Aktivitäten“ (TLAs)
  - Jede Aktion im Frontend hat eine zugeordnete TLA
  - Orientierung am Command-Pattern
  - Klar definierter Input/Output
  - Fehler/Ausnahmen als Fehler-Code innerhalb des Outputs

Top-Aktiv im Frontend

```
@Stateless
@Local(AntragErzeugenActivity.class)
@Name("AntragErzeugenActivity")
@LocalBinding(jndiBinding = "AntragErzeugenActivityImpl/local")
@TransactionManagement(TransactionManagementType.BEAN)
public class AntragErzeugenActivityImpl extends
AbstractStateOnlyProcessActivity<List<Kerndatensatz>, GesamtAntrag>
    implements AntragErzeugenActivity, Serializable {
    @EJB
    private AntragstellerZuGesamtantragHinzufuegenActivity
                                          antragstellerHinzufuegenActivity;
    @EJB
    private ProzessProfilInformationenLadenActivity profilDatenLadenActivity;

    @In(value = "AntragsProcessHolder", create = true)
    @Out(value = "AntragsProcessHolder")
    private AntragsProcessHolder prozess;

    @Logger
    Log log;

    @Override
    protected ActivityResult<List<Kerndatensatz>, Void>
                    executeStep(List<Kerndatensatz> kerndatensaetze) {
      ...
    }
}
```

Das Rad nicht immer neu erfinden

- Immer wiederkehrende Abläufe in verschiedenen Kontexten
  - „Basis Aktivitäten“ (BAs) als wiederverwendbare Bausteine
  - TLAs nutzen i.d.R. BAs zur Realisierung ihrer Aufgaben
  - In Ausnahmen auch direkter Durchgriff auf die Backend-Komponenten möglich
    (Schichten-Reduzierung)

Das heilige Datenmodell

- Front- und Backend nutzen das gleiche (aber NICHT: das selbe!) Datenmodell
  - Kopie der Daten im Frontend (JPA-Entitäten als Seam-Beans)
  - „Daten-Master“ wird im Backend (TLA) gehalten
  - Validierung der Daten erfolgt im Backend
    - Gewährleistung der Konsistenz des Masters
    - Kein Arbeiten auf „Attached Entities“

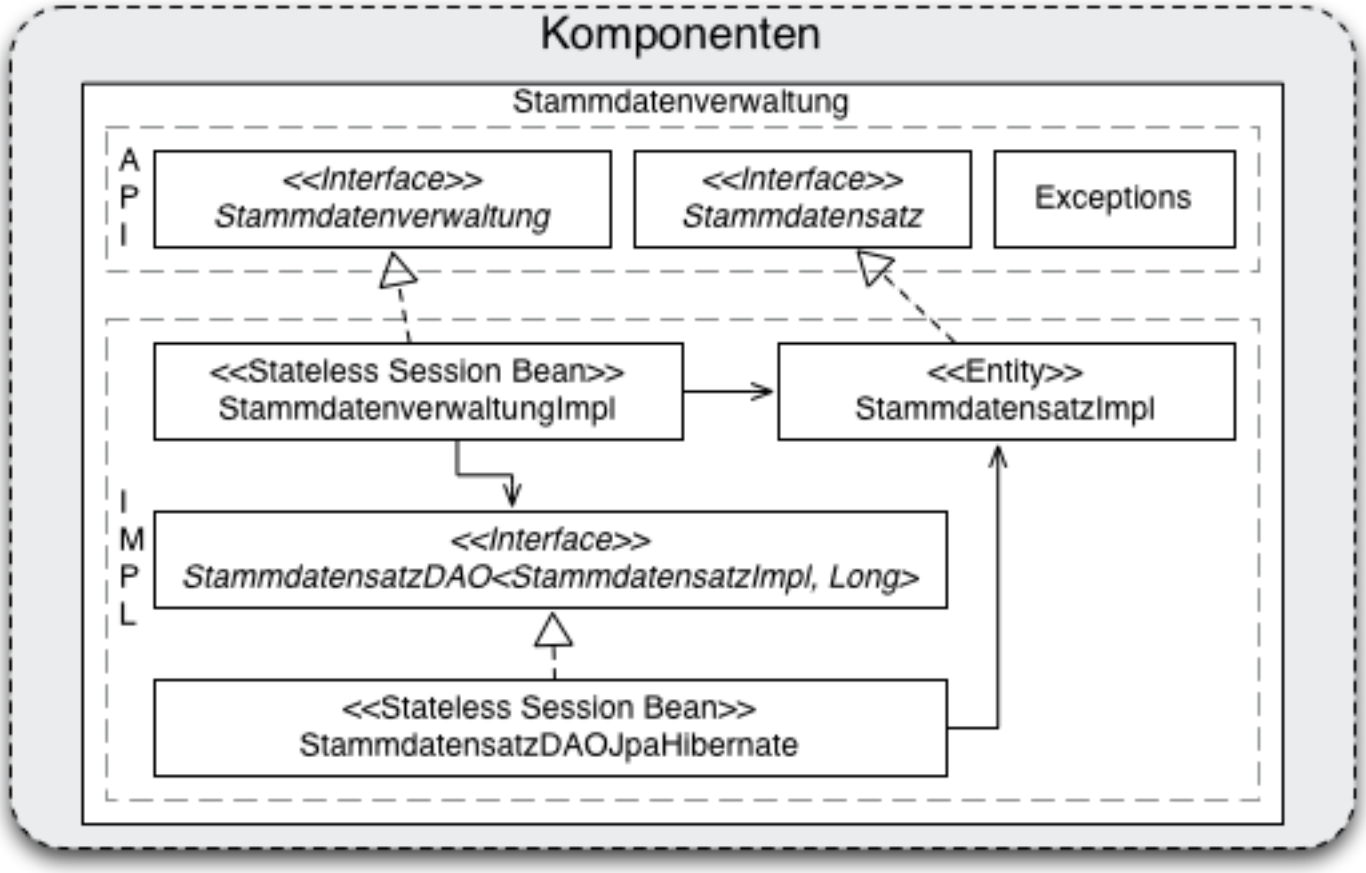
Die Sache mit den Transaktionen

- Einsatz von zustandsbehafteten TLAs
  - Transienter Zustand
  - Muss Teil der aktiven Transaktion sein
- Mögliche Lösungen:
  - SessionSynchronization
  - Aktive Transaktionssteuerung (BMT)

Die Sache mit den Transaktionen

- Die Wahl viel in diesem Fall auf BMT
  - Aktive Transaktionssteuerung in den TLAs
  - Bereitstellung abstrakter Basisklassen
    - Realisierung von Objekt-Transaktionen
    - Kein Mehraufwand in konkreten Implementierungen
  - Transaktionsattribut MANDATORY für BAs (CMT)

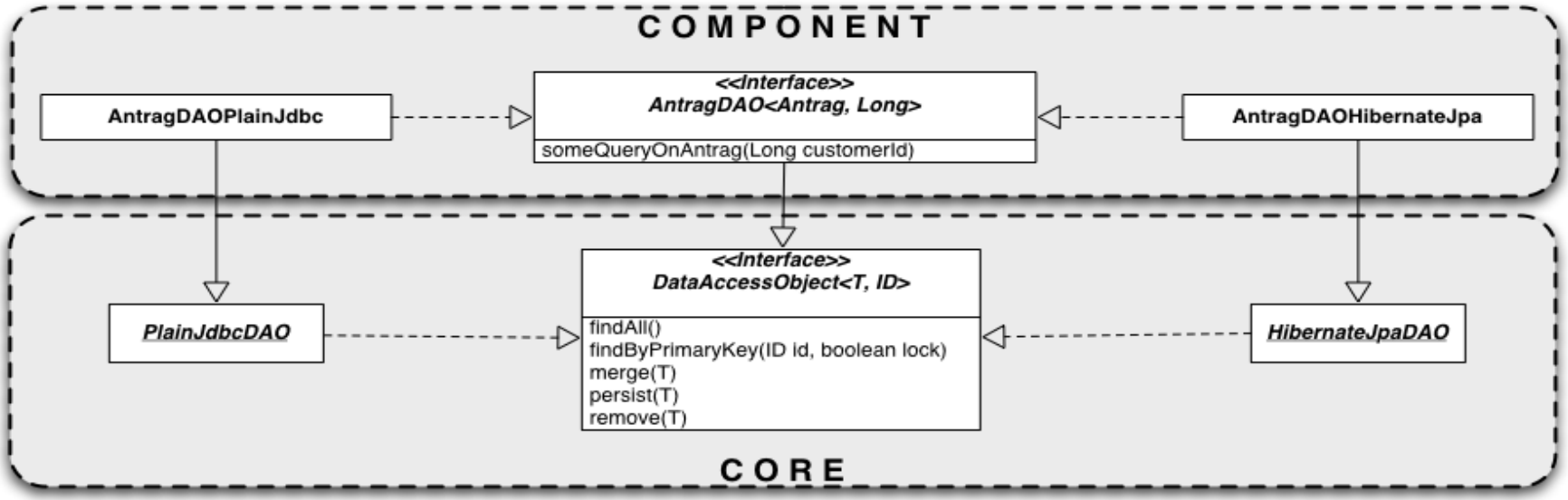
Architektur der Komponenten



Architektur der Komponenten

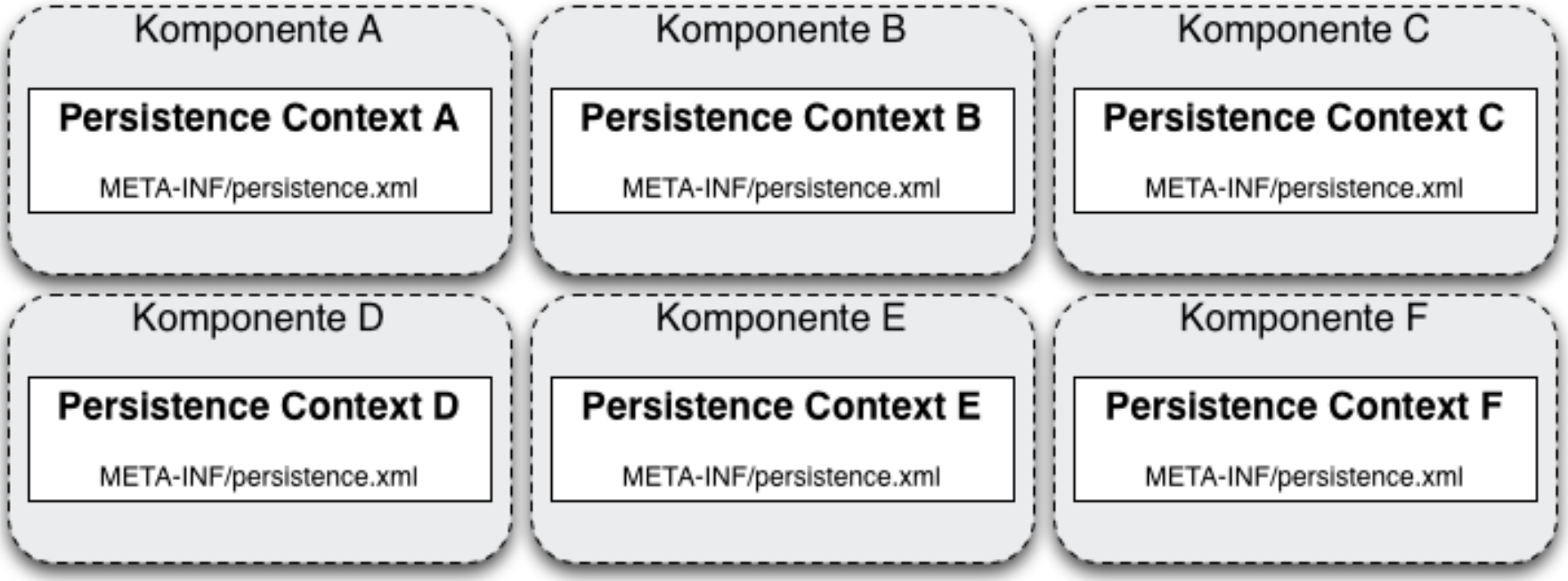
- Keine annotierten Klassen in API Projekt
- TransactionAttribute.MANDATORY
- EJB3 basierte Lösung (everything an EJB)
- Basis Komponenten referenzieren sich nicht gegenseitig
- Andere Projekte dürfen nur das API Package referenzieren

Persistenz Layer in einer Komponente



Segmentierung von Persistence Units

- Ausserhalb einer Komponente sind Entitäten detached



Persistenz

- Verwendung von Java Persistence API
- Hibernate als Persistence Provider
  - Hibernate Core
  - Hibernate Annotations
  - Hibernate Entity Manager
- Fallback auf „Plain Hibernate“ ist erlaubt
- Keine XML Mappings, nur Annotationen

Wie weit kommt man mit JPA?

Erstaunlicherweise: sehr weit

Verwendete Vendor Extensions

- Mappings
  - Collections of Components
  - Fetch Types
  - Caching
- API
  - Criteria Queries
  - saveOrUpdate

Caching

- Default war deaktivierter Second Level Cache
- Bei Bedarf wurde ein Process Level Second Level Cache aktiviert
- Cache Provider EHCache

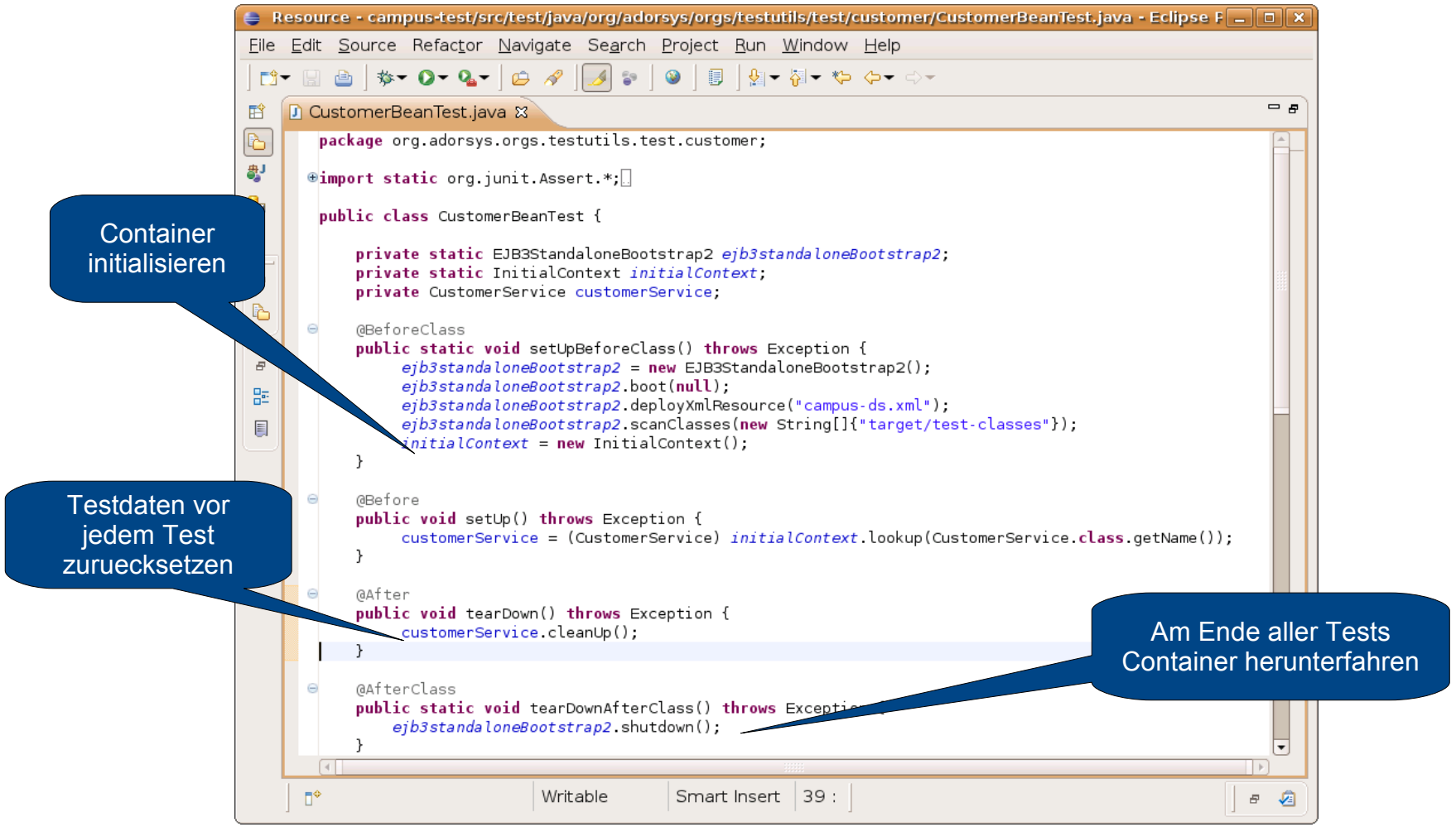
Die Entwicklungsumgebung

- Schneller Roundtrip bei der Entwicklung einzelner Use-Cases.
- Erhöhung der Qualität und Quantität von Unit-Tests.
- Bessere Kontrolle über eingesetzte bzw. benötigte fremde Komponenten.
- Integrierte Dokumentation.

Maven2 als Build-Tool

- Automatisches Dependency Management
- Fast perfekte Integration mit eclipe und idea
  - mvn eclipse:eclipse, mvn idea:idea
  - Leider zum Projektstart noch keine vernünftige WTP-Integration.
- IDE-Artefakte werden nicht eingechecked
  - .classpath, .project
- ANT-Unterstützung für Deployment

Test mit JBoss Embedded EJB3



Vorteil des Embedded Container

- Sehr hohe Qualität bei geringem Einsatz
  - Test unter realen Bedingungen, Annotationen, Transaktionen, Bean-Interceptors, schnelles Debugging.
  - Man kann mehr Tests schreiben, man kommt schneller voran.
  - Man hat in den späteren Phasen nicht nur mit weniger Bugs zu kämpfen, sondern kann entdeckte Fehler schnell und zuverlässig beheben.

Nachteile des Embedded Container

- Einstiegshürde immer noch zu hoch.
- Bietet nur eingeschränkte Funktionalität (jedoch das Wesentliche).
- Integration in die Front-End Entwicklung nicht zufriedenstellend.

Ausblick

- JBoss bietet mit der 5.0 Reihe einen besser abgestimmten Embedded Container.
- Andere Anbieter folgen den Weg:
  - OpenEJB 3.x
  - Pitchfork
  - Glassfisch V3
- Und langsam wird die Maven-Eclipse WST Integration immer besser.

Lessons learned (Seam)

- Seam verspricht viel
  - Nicht alles ist Gold
    - Mehrschicht-Systeme existieren nicht ohne Grund
    - Getrennte Datenhaltung (FE/BE) bleibt wichtiges Fangnetz
    - Kein Zugriff auf Remote-EJBs
    - Undurchsichtige Defaults (Bsp.: JNDI-Namen)
    - Mangelnde Dokumentation
    - Erschwertes Debugging
    - Sehr empfindlich bzgl. Third-Party-Bibliotheken (Versionsproblematik)

Lessons learned (Seam)

- Seam verspricht viel
  - Aber viele Erleichterungen
    - Komfortable FrontendBean-Organisation durch Deklaration mit Annotationen und In-/Outjection (Bijection)
    - Bessere Navigationsstrukturierung in der pages.xml
    - Preaction-Definition möglich
    - Erweiterte Expression-Language
- Es hat eine Menge Schweiß gekostet, aber es funktioniert !

Lessons learned (JPA/EJB3)

- JPA ist durchdacht
- Integration in das EJB3 Programmiermodell ist hervorragend
- Insbesondere Collections of Components fehlen
- Integration in JBoss war problemlos möglich

Vielen Dank!

Werner Eberling / Francis Pouatcha

MATHEMA Software GmbH / adorsys Ltd. & Co. KG